A Scalable Implementation of a MapReduce-based Graph Processing Algorithm for Large-scale Heterogeneous Supercomputers

Koichi Shirahata[*1], Hitoshi Sato[*1,*2],
Toyotaro Suzumura[*1,*2,*3], Satoshi Matsuoka[*1]

[*1] Tokyo Institute of Technology

[*2] CREST, Japan Science and Technology Agency

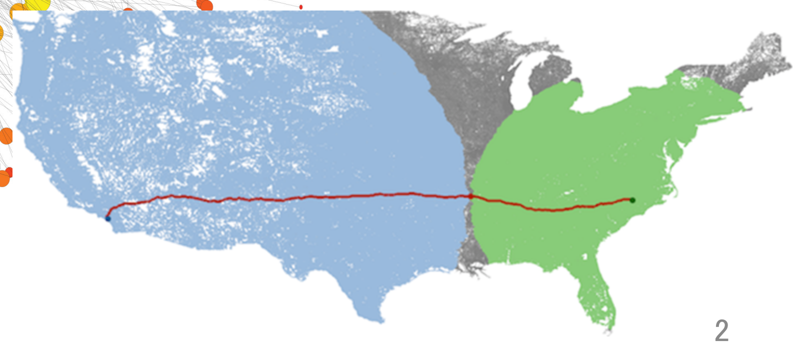
[*3] IBM Research - Tokyo

Emergence of Large Scale Graphs

⇨ **Need fast and scalable analysis using HPC**

facebook

900 Million Vertices
100 Billion Edges



GPU-based Heterogeneous supercomputers

TSUBAME 2.0

1408 compute nodes (3 GPUs / node)

GPGPU

High peak performance

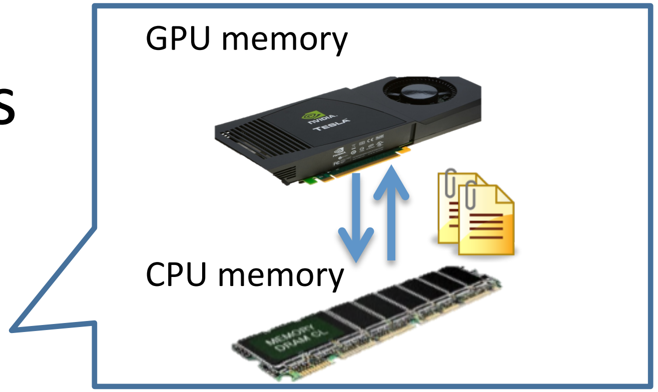
High memory bandwidth

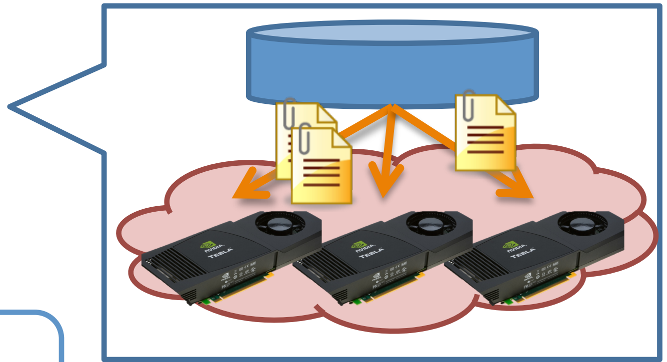
Motivation

Fast Large Graph Processing with GPGPU

Problems of Large Scale Graph Processing with GPGPU

- **How much do GPUs accelerate large scale graph processing ?**
  - Applicability to graph applications
    - Computation patterns of graph algorithm affects performance
    - Tradeoff between computation and CPU-GPU data transfer overhead
  - How to distribute graph data to each GPU in order to exploit multiple GPUs





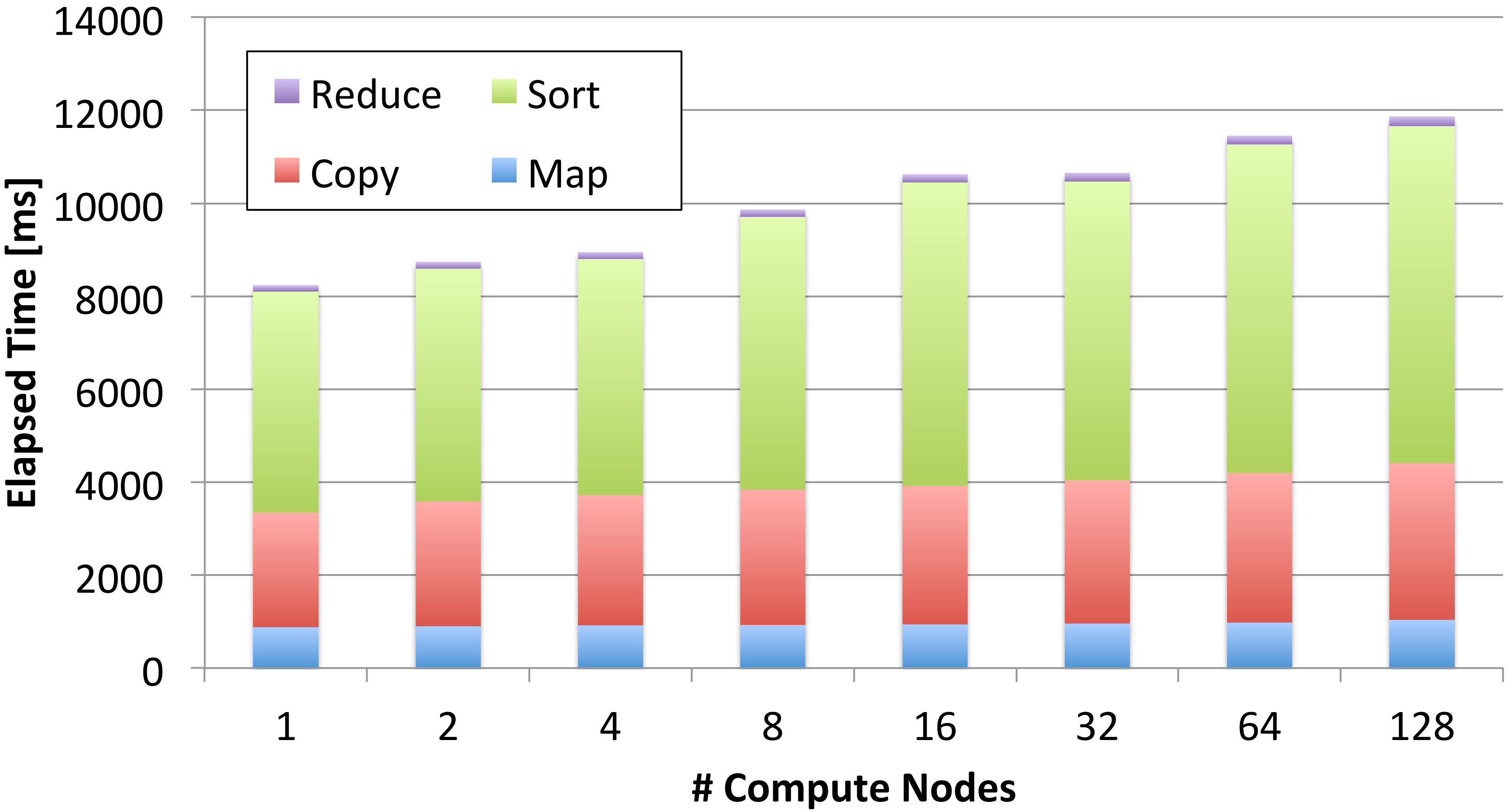
Scalability | Load balancing | Communication

Motivating Example: CPU-based Graph Processing

- **How much is the graph application accelerated using GPU ?**


Simple computation patterns, High memory bandwidth


Complex computation patterns, PCI-E overhead



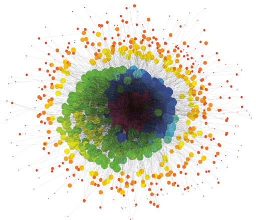
Contributions

- **Implemented a scalable multi-GPU-based PageRank application**
  - Extend Mars (an existing GPU MapReduce framework)
    - Using the MPI library
  - Implement GIM-V on multi-GPU MapReduce
    - GIM-V: a graph processing algorithm
  - Load balance optimization between GPU devices for large-scale graphs
    - Task scheduling-based graph partitioning

Performance on TSUBAME2.0 supercomputer

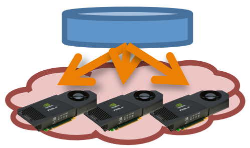
- **Scale well up to 256 nodes (768 GPUs)**
- **1.52x speedup compared with on CPUs**

Proposal: Multi-GPU GIM-V with Load Balance Optimization



Graph Application

PageRank

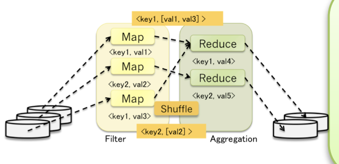


Graph Algorithm

Multi-GPU GIM-V

Implement GIM-V on multi-GPUs MapReduce

- Optimization for GIM-V
- Load balance optimization



MapReduce Framework

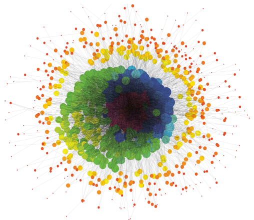
Multi-GPU Mars

Extend an existing GPU MapReduce framework (Mars) for multi-GPU

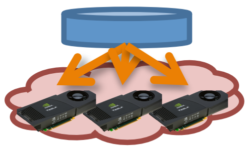
Platform

CUDA, MPI

Proposal: Multi-GPU GIM-V with Load Balance Optimization

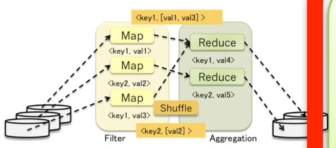


Graph Application
PageRank



Graph Algorithm
Multi-GPU GIM-V

Implement GIM-V on multi-GPUs MapReduce
- Optimization for GIM-V
- Load balance optimization



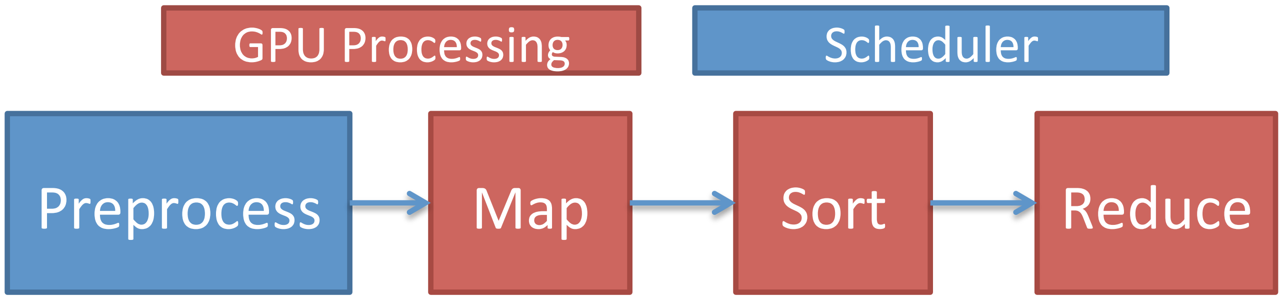
MapReduce Framework
Multi-GPU Mars

Extend an existing GPU MapReduce framework (Mars) for multi-GPU

Platform
CUDA, MPI

Structure of Mars

- Mars*[1] : an existing GPU-based MapReduce framework
  - CPU-GPU data transfer (Map)
  - GPU-based Bitonic Sort (Shuffle)
  - Allocates one CUDA thread / key (Map, Reduce)

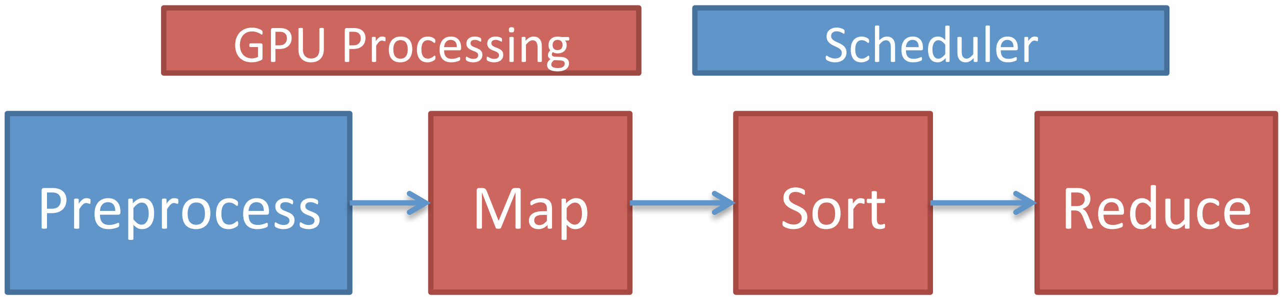


*1 : Bingsheng He et al. Mars: A MapReduce Framework on Graphics Processors. PACT 2008

Structure of Mars

- Mars*[1] : an existing GPU-based MapReduce framework
  - CPU-GPU data transfer (Map)
  - GPU-based Bitonic Sort (Shuffle)
  - Allocates one CUDA thread / key (Map, Reduce)

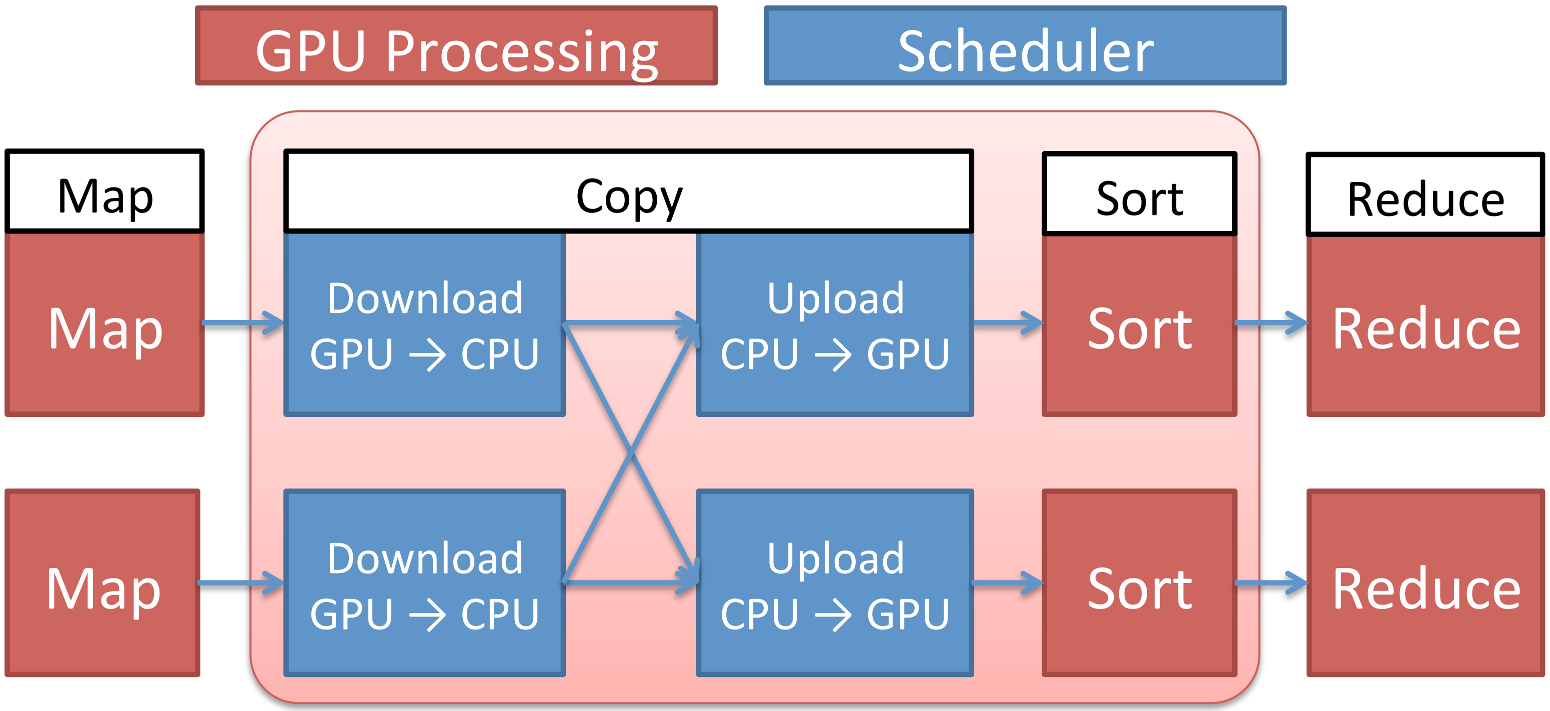
→ **We extend Mars for multi-GPU support**



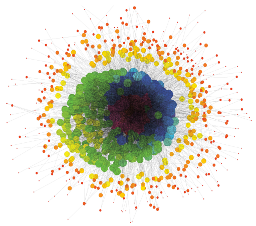
*1 : Bingsheng He et al. Mars: A MapReduce Framework on Graphics Processors. PACT 2008

Proposal:
Mars Extension for Multi-GPU using MPI

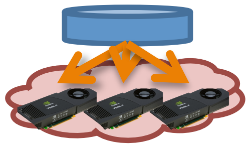
- Inter-GPU communications in Shuffle
  - G2C → MPI_Alltoallv → C2G → local Sort
- Parallel I/O feature using MPI-IO
  - Improve I/O throughput between memory and storage



Proposal: Multi-GPU GIM-V with Load Balance Optimization

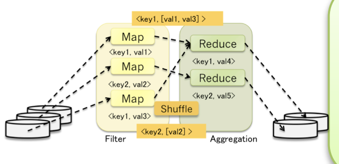


Graph Application
PageRank



Graph Algorithm
Multi-GPU GIM-V

Implement GIM-V on multi-GPUs MapReduce
- Optimization for GIM-V
- Load balance optimization



MapReduce Framework
Multi-GPU Mars

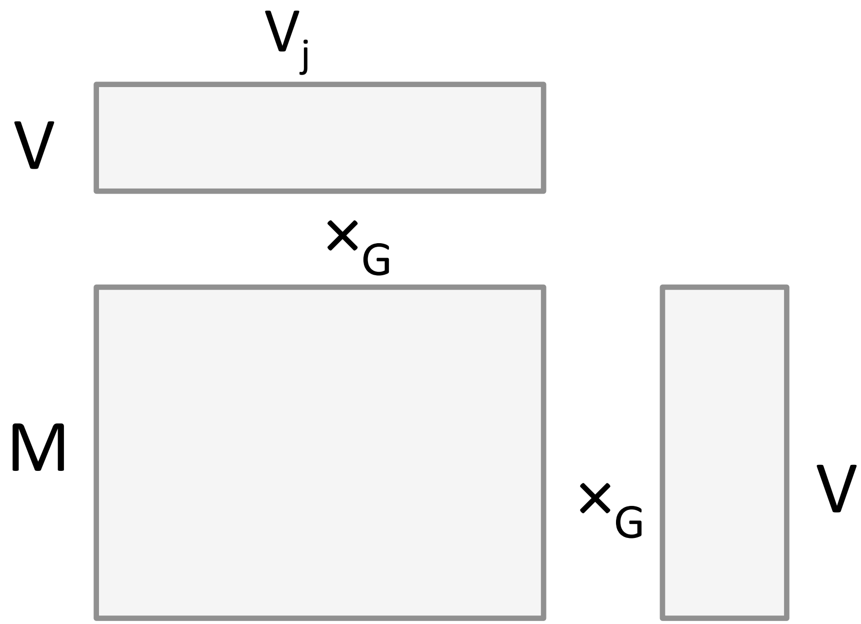
Extend an existing GPU MapReduce framework (Mars) for multi-GPU

Platform
CUDA, MPI

Large graph processing algorithm GIM-V

- **Generalized Iterative Matrix-Vector multiplication**[*1]
  - Graph applications are implemented by defining 3 functions
  - $v' = M \times_G v$ where

    v'_i = Assign(v_j , CombineAll$_j$ ({x_j | j = 1..n, x_j = Combine2($m_{i,j}$, v_j)})) (i = 1..n)

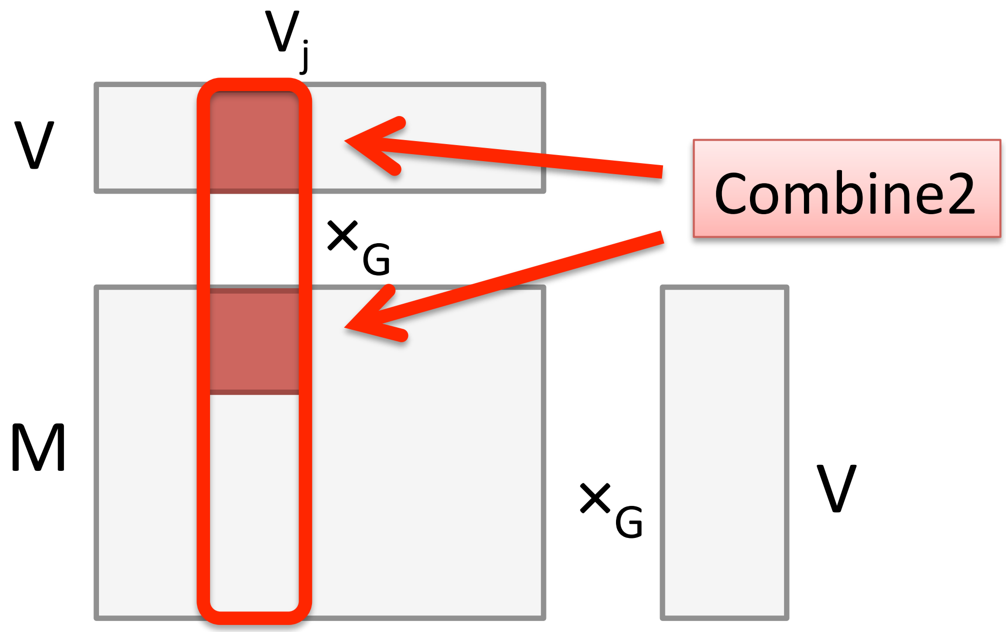


*1 : Kang, U. et al, "PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations", IEEE INTERNATIONAL CONFERENCE ON DATA MINING 2009

Large graph processing algorithm GIM-V

- **Generalized Iterative Matrix-Vector multiplication***1
  - Graph applications are implemented by defining 3 functions
  - $v' = M \times_G v$ where

    v'_i = Assign(v_j , CombineAll$_j$ ($\{x_j \mid j = 1..n, x_j$ = Combine2($m_{i,j}$, v_j)$\}$)) (i = 1..n)

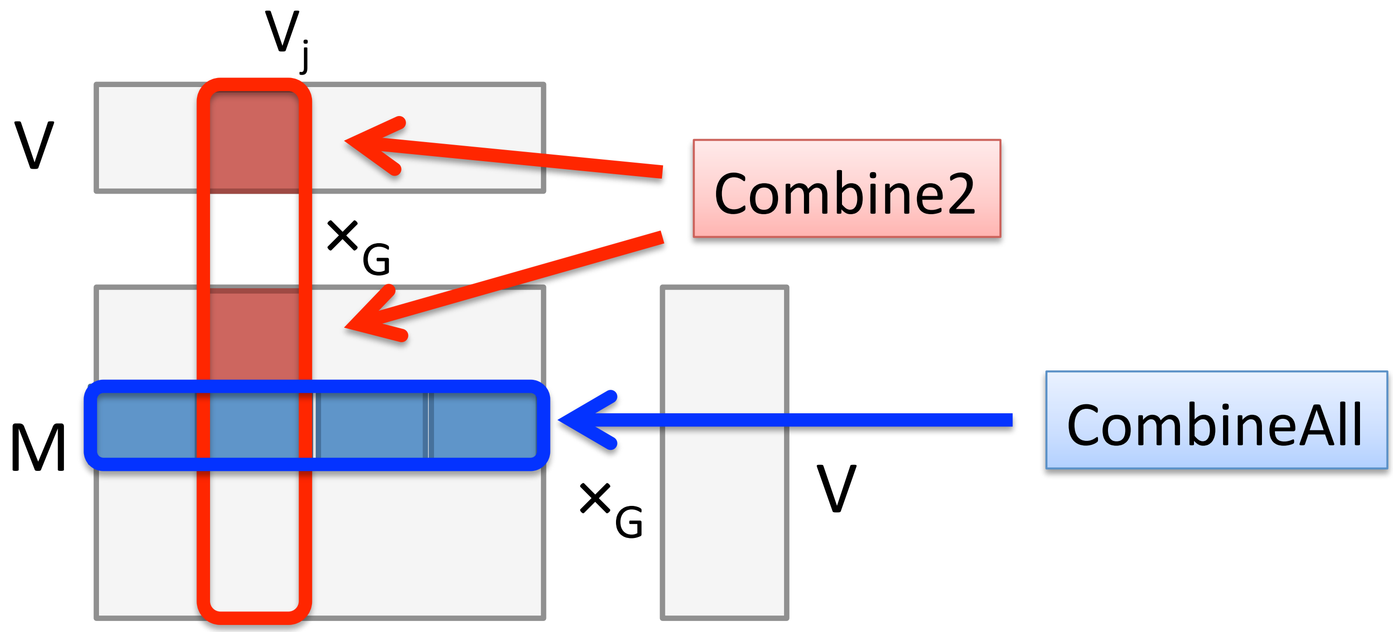


*1 : Kang, U. et al, "PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations", IEEE INTERNATIONAL CONFERENCE ON DATA MINING 2009

Large graph processing algorithm GIM-V

- **Generalized Iterative Matrix-Vector multiplication***1
  - Graph applications are implemented by defining 3 functions
  - $v' = M \times_G v$ where

    $v'_i = \text{Assign}(v_j, \text{CombineAll}_j(\{x_j \mid j = 1..n, x_j = \text{Combine2}(m_{i,j}, v_j)\}))$ $(i = 1..n)$

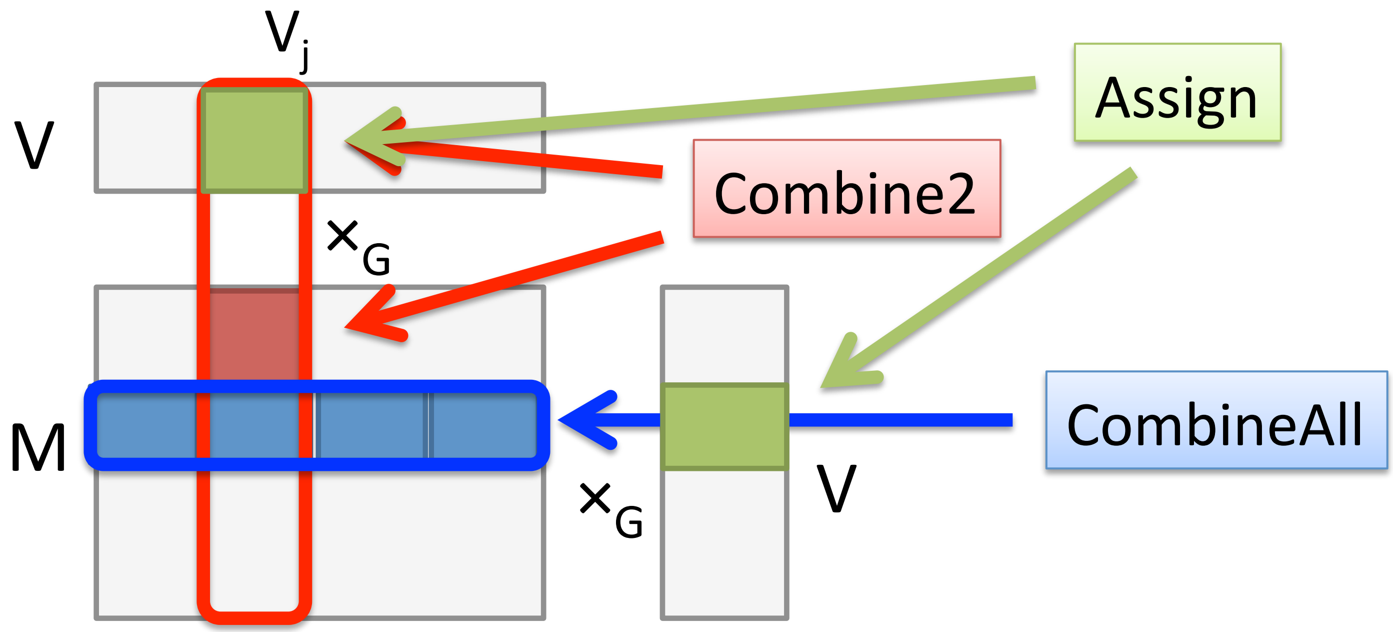


*1 : Kang, U. et al, "PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations", IEEE INTERNATIONAL CONFERENCE ON DATA MINING 2009

Large graph processing algorithm GIM-V

- **Generalized Iterative Matrix-Vector multiplication**[*1]
  - Graph applications are implemented by defining 3 functions
  - $v' = M \times_G v$ where

    $v'_i = \text{Assign}(v_j, \text{CombineAll}_j(\{x_j \mid j = 1..n, x_j = \text{Combine2}(m_{i,j}, v_j)\}))$ $(i = 1..n)$



*1 : Kang, U. et al, "PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations", IEEE INTERNATIONAL CONFERENCE ON DATA MINING 2009

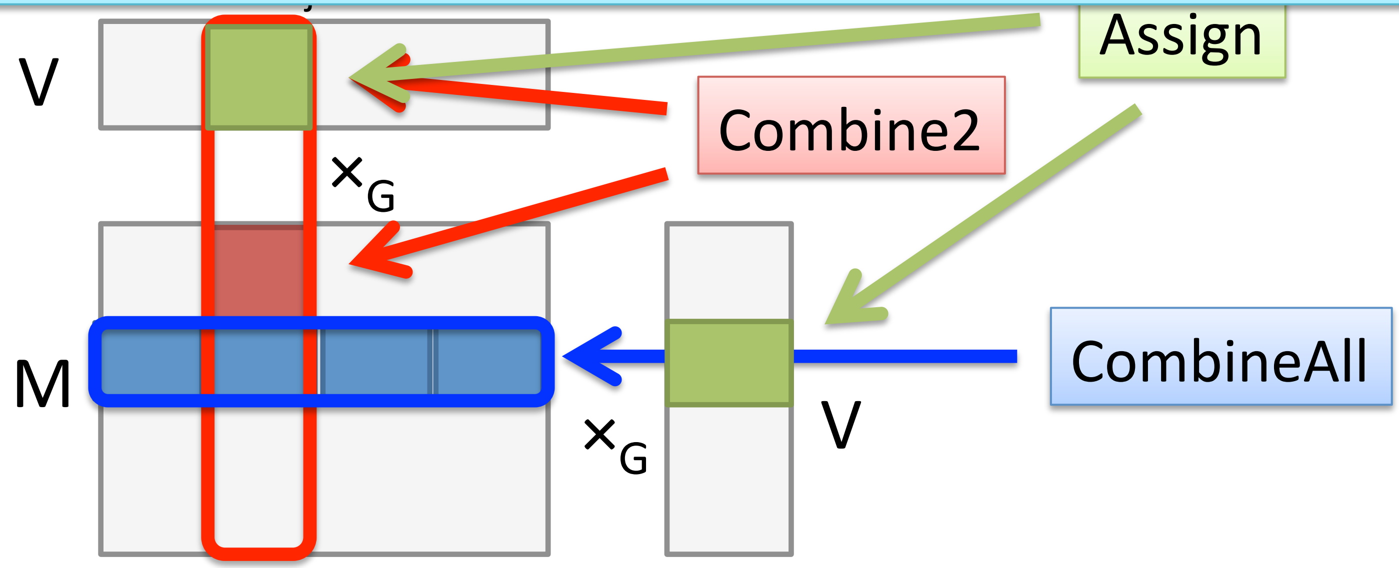
Large graph processing algorithm GIM-V

- **Generalized Iterative Matrix-Vector multiplication**[*1]
  - Graph applications are implemented by defining 3 functions
  - $v' = M \times_G v$ where

    $v'_i = \text{Assign}(v_i, \text{CombineAll}_i(\{x_j \mid j = 1..n, x_j = \text{Combine2}(m_{i,j}, v_j)\}))$ $(i = 1..n)$

GIM-V can be implemented by 2-stage MapReduce

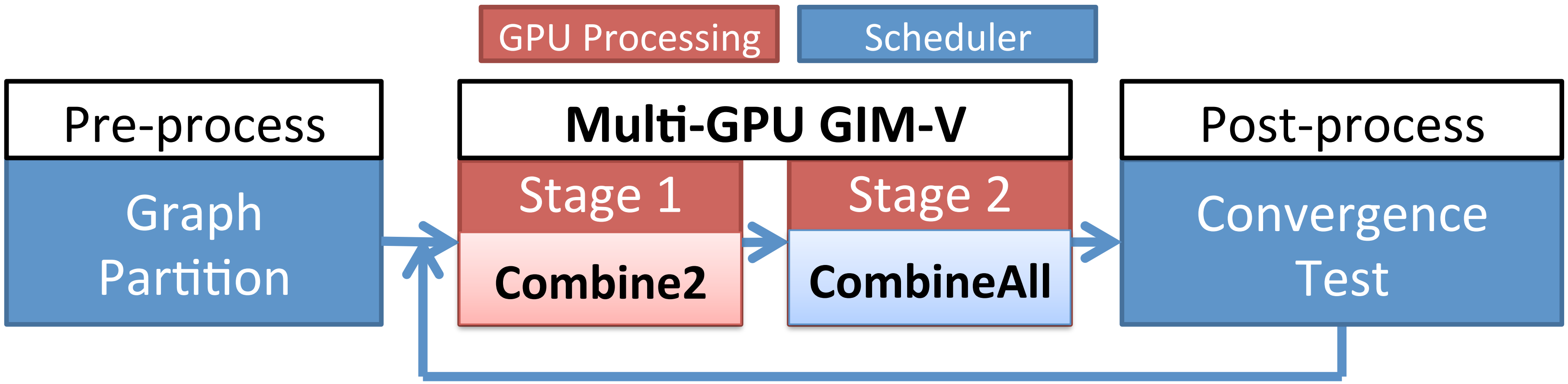
→ Implement on multi-GPU environment



*1 : Kang, U. et al, "PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations", IEEE INTERNATIONAL CONFERENCE ON DATA MINING 2009

Proposal:
GIM-V implementation on multi-GPU

- **Continuous execution feature for iterations**
  - 2 MapReduce stages / iteration
  - Graph partition at Pre-processing
    - Divide the input graph vertices/edges among GPUs
  - Parallel Convergence test at Post-processing
    - Locally on each process -> globally using MPI_Allreduce



Optimizations for multi-GPU GIM-V

Mars

- **Data structure**
  - Mars handles metadata and payload
- **Thread allocation**
  - Mars handles one key per thread
- **Load balance optimization**
  - Scale-free property
    - Small number of vertices have many edges

Our Implementation

Eliminate metadata and use fixed size payload

In Reduce stage, **allocate multi CUDA threads to a single key** according to value size

Minimize load imbalance among GPUS

Optimizations for multi-GPU GIM-V

Mars

- **Data structure**
  - Mars handles <u>metadata and payload</u>
- **Thread allocation**
  - Mars handles <u>one key per thread</u>
- **Load balance optimization**
  - Scale-free property
    - <u>Small number of vertices have many edges</u>

Our Implementation

Eliminate metadata and use fixed size payload

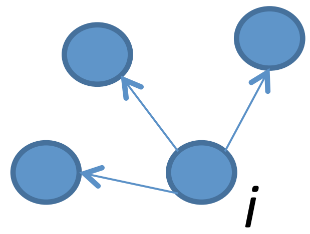
In Reduce stage, **allocate multi CUDA threads to a single key** according to value size

Minimize load imbalance among GPUS

Apply Load Balancing Optimization

- **Partition the graph in order to minimize load imbalance among GPUs**
  - Applying a task scheduling algorithm
    - Regard Vertex/Edges as Task
    - $\text{TaskSize}_i = 1 + \Sigma$ Outgoing Edges

Vertex $_i$



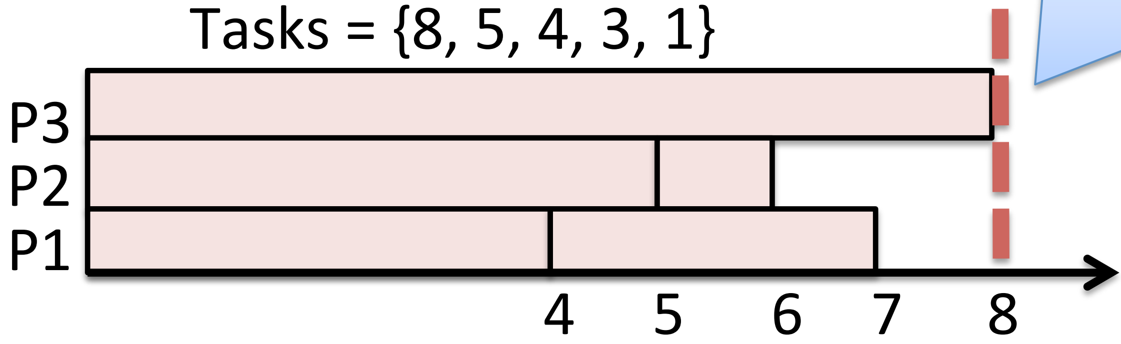
$\text{TaskSize}_i = 1 + 3$

V E$_{out}$

  - LPT (Least Processing Time) schedule *1
    - Assign tasks in decreasing order of task size

Minimize the maximum amount

Tasks = {8, 5, 4, 3, 1}



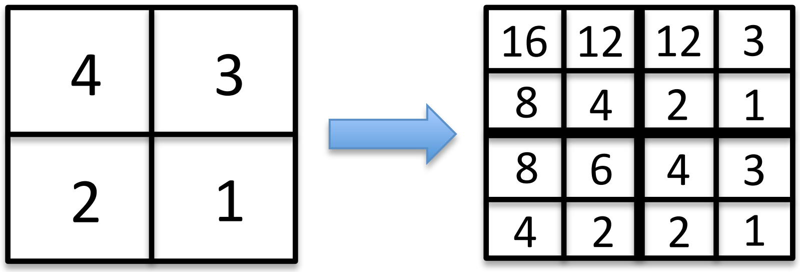
*1 : R. L. Graham, "Bounds on multiprocessing anomalies and related packing algorithms," in *Proceedings of the May 16-18, 1972, spring joint computer conference,* ser. AFIPS '72 (Spring)

Experiments

Study the performance of our multi-GPU GIM-V

- **Scalability**
- **Comparison w/ a CPU-based implementation**
- **Validity of the load balance optimization**

- Methods
  - A single round of iterations (w/o Preprocessing)
  - PageRank application
    - Measures relative importance of web pages
  - Input data
    - Artificial Kronecker graphs
      - Generated by generator in Graph 500
    - Parameters
      - *SCALE*: log 2 of #vertices (#vertices = 2^{SCALE})
      - *Edge_factor*: 16 (#edges = *Edge_factor* × #vertices)



G_1 $\quad$ $G_2 = G_1 \otimes G_1$

Experimental environments

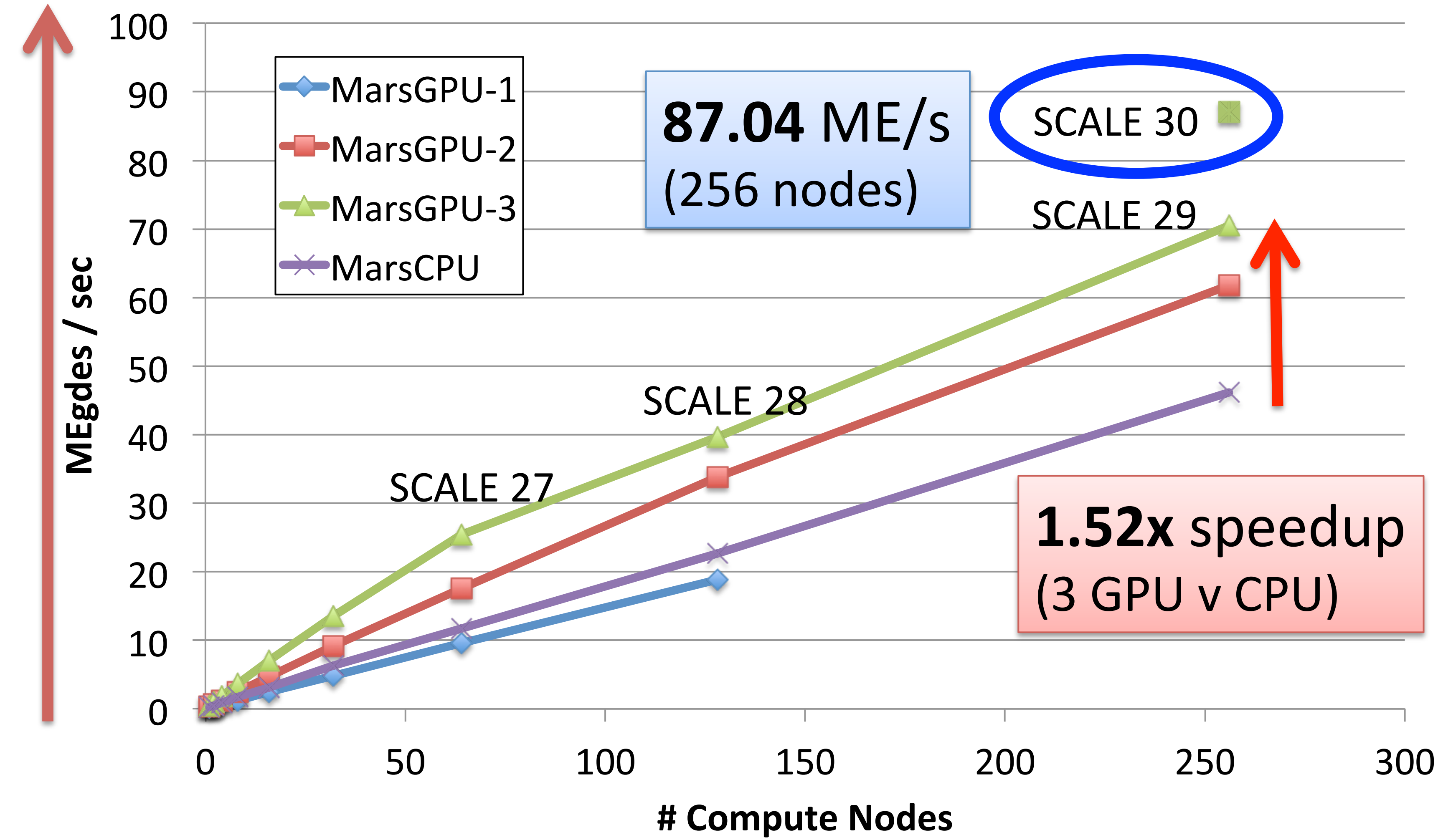
- TSUBAME 2.0 supercomputer
  - We use 256 nodes (768 GPUs)
    - CPU-GPU: PCI-E 2.0 x16
    - Internode: QDR IB (40 Gbps) dual rail

- Mars
  - *MarsGPU-n*
    - n GPUs / node (n: 1, 2, 3)
  - *MarsCPU*
    - 12 threads / node
    - MPI and pthread
    - Parallel quick sort

	CPU	GPU
Model	Intel® Xeon® X5670	Tesla M2050
# Cores	6	448
Frequency	2.93 GHz	1.15 GHz
Memory	54 GB	2.7 GB
Compiler	gcc 4.3.4	nvcc 4.0

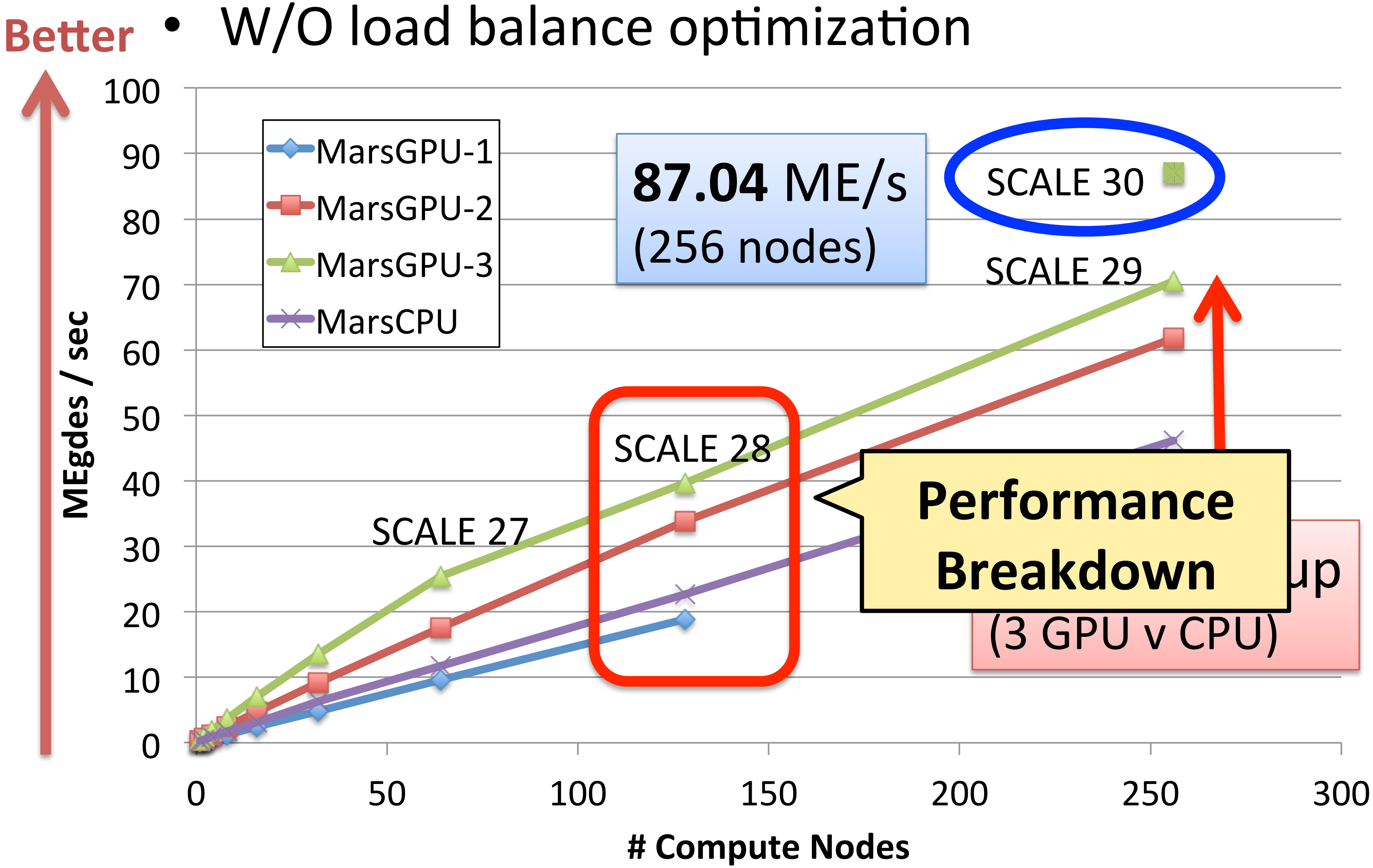
Weak Scaling Performance: *MarsGPU* vs. *MarsCPU*

- W/O load balance optimization

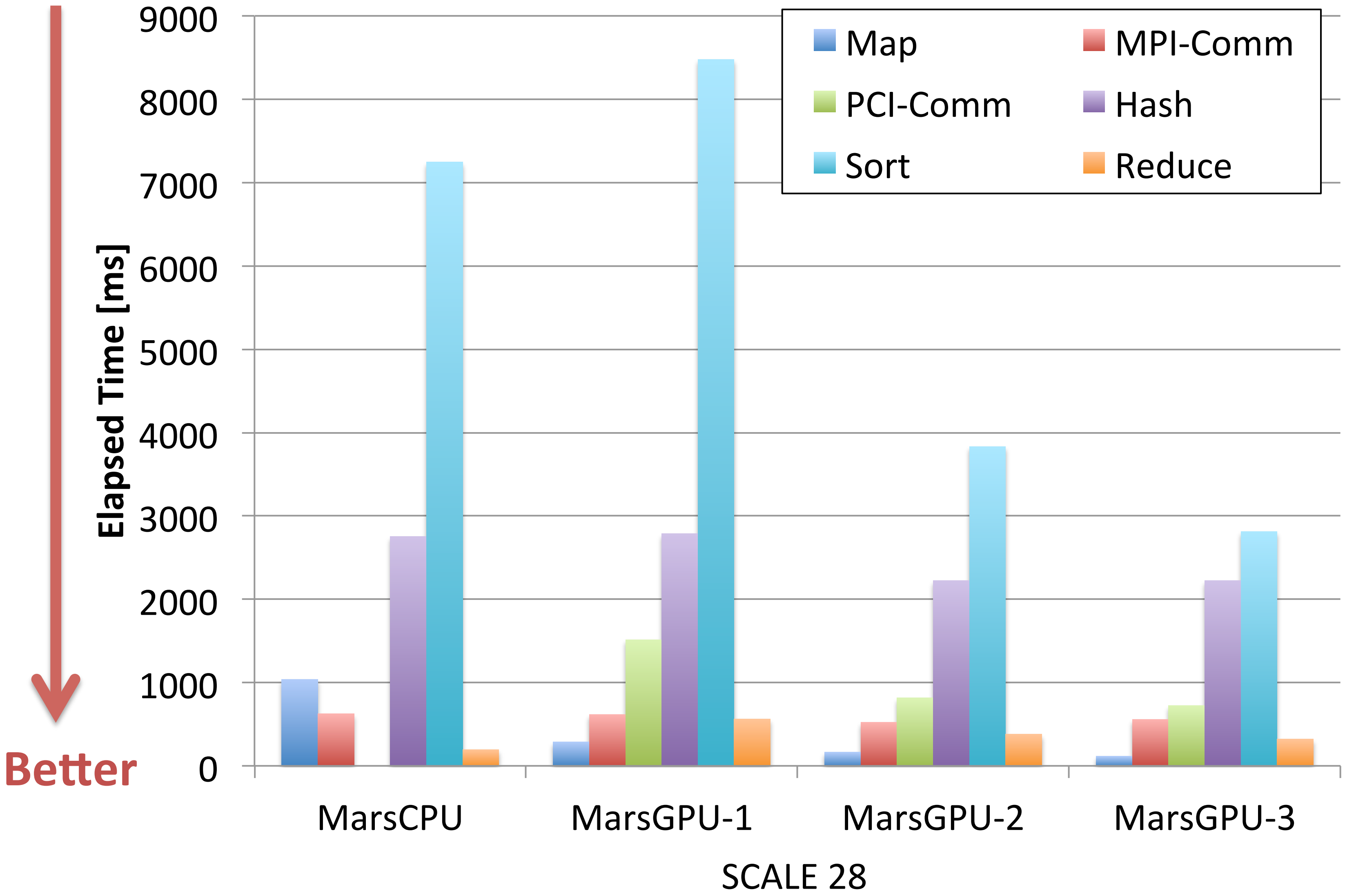


Weak Scaling Performance: *MarsGPU* vs. *MarsCPU*

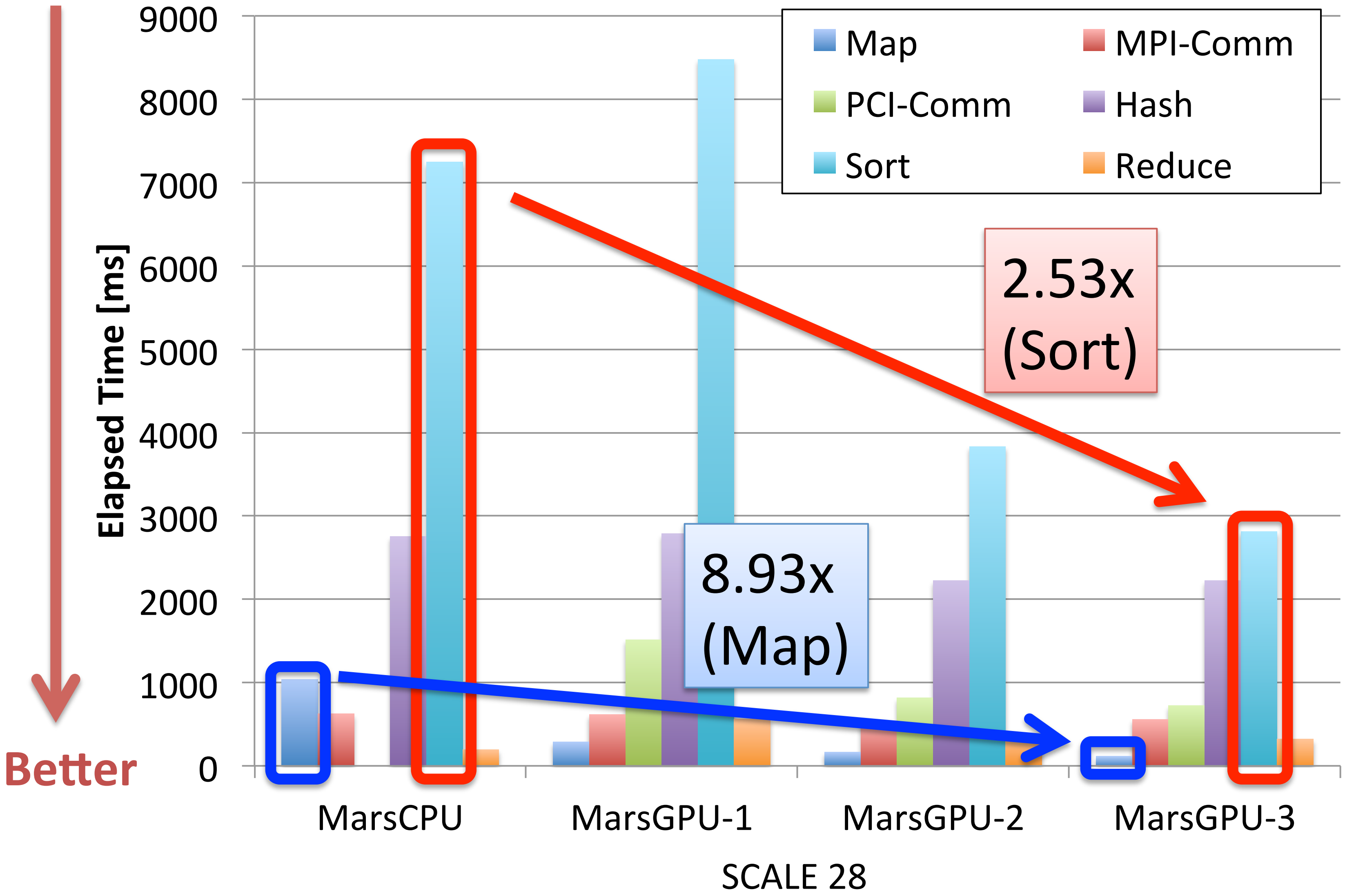
- W/O load balance optimization



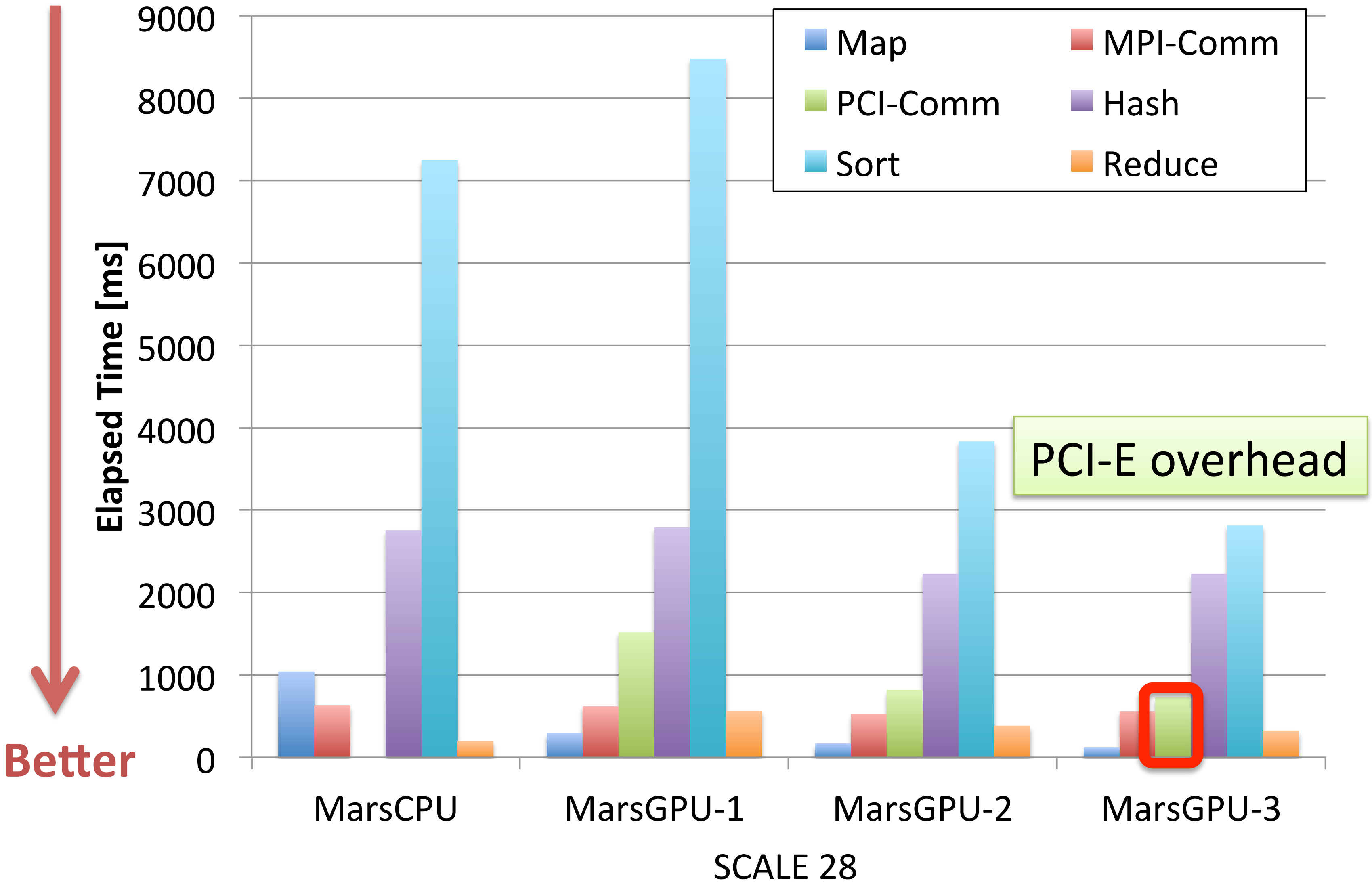
Performance Breakdown: *MarsGPU* and *MarsCPU*



Performance Breakdown: *MarsGPU* and *MarsCPU*

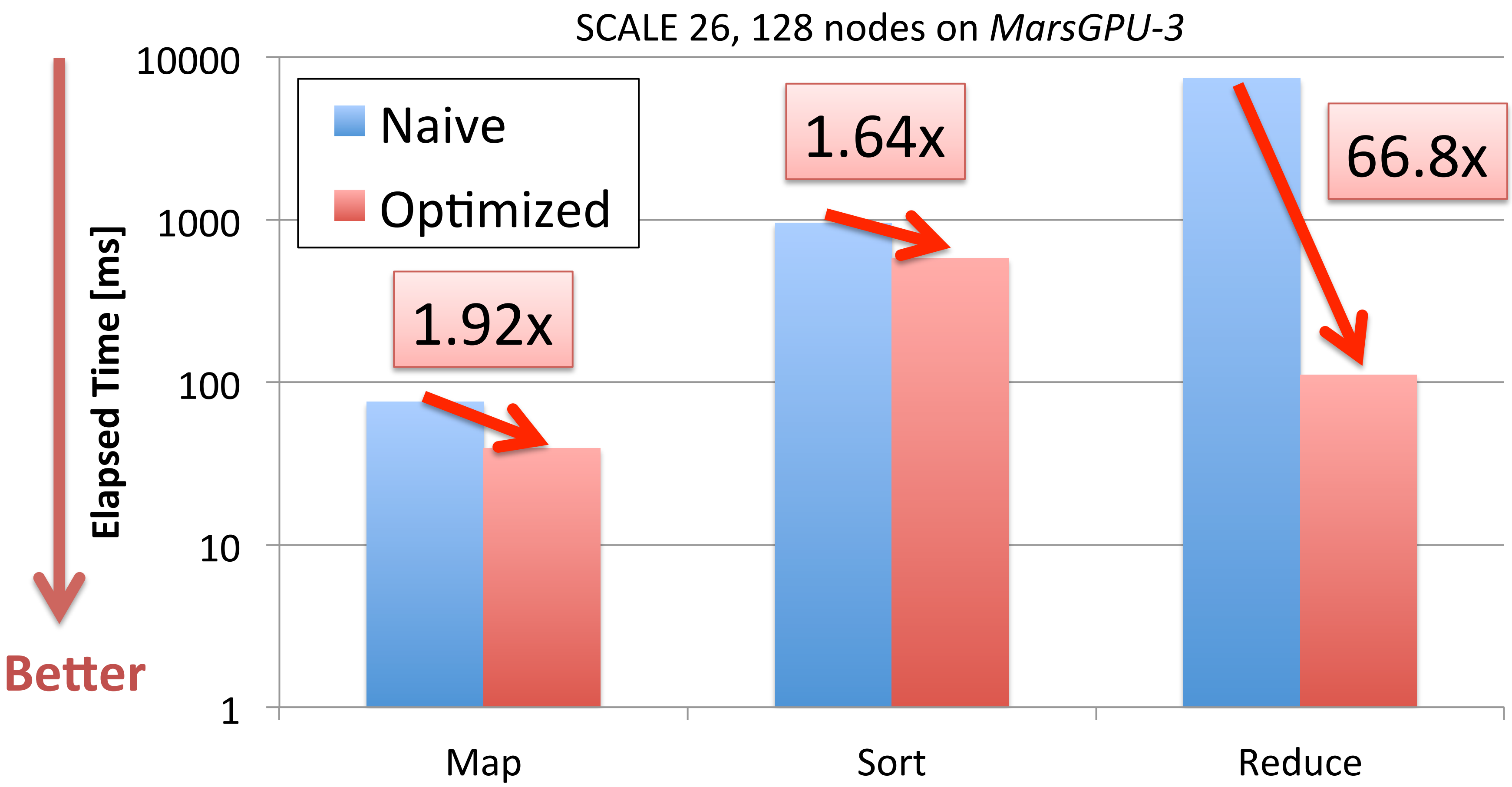


Performance Breakdown: *MarsGPU* and *MarsCPU*



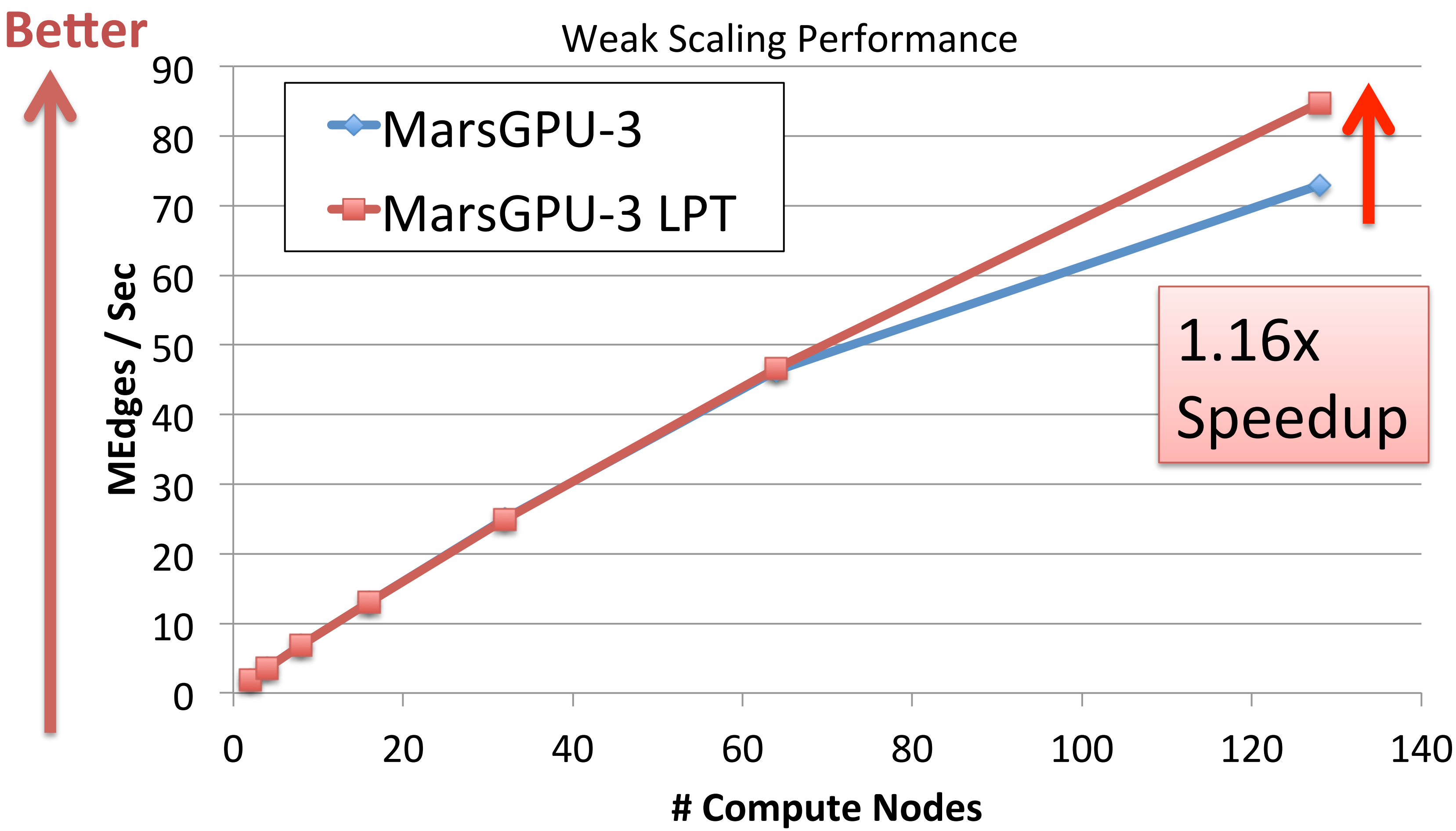
Efficiency of GIM-V Optimizations

- **Data structure** (Map, Sort, Reduce)
- **Thread allocation** (Reduce)



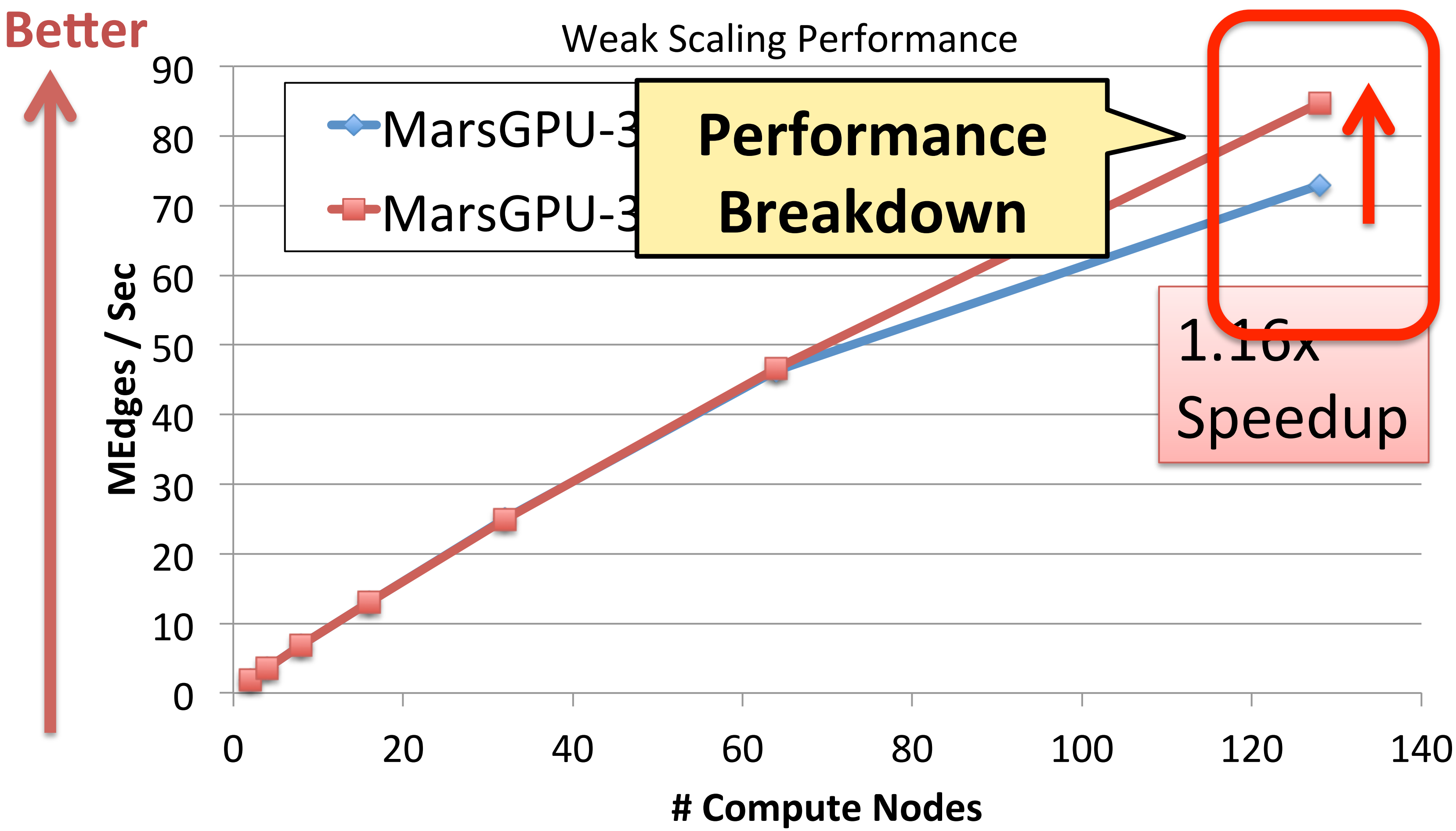
Round Robin vs. LPT Schedule

- Similar except for on 128 nodes
  - Input graphs are relatively well-balanced (Graph500)



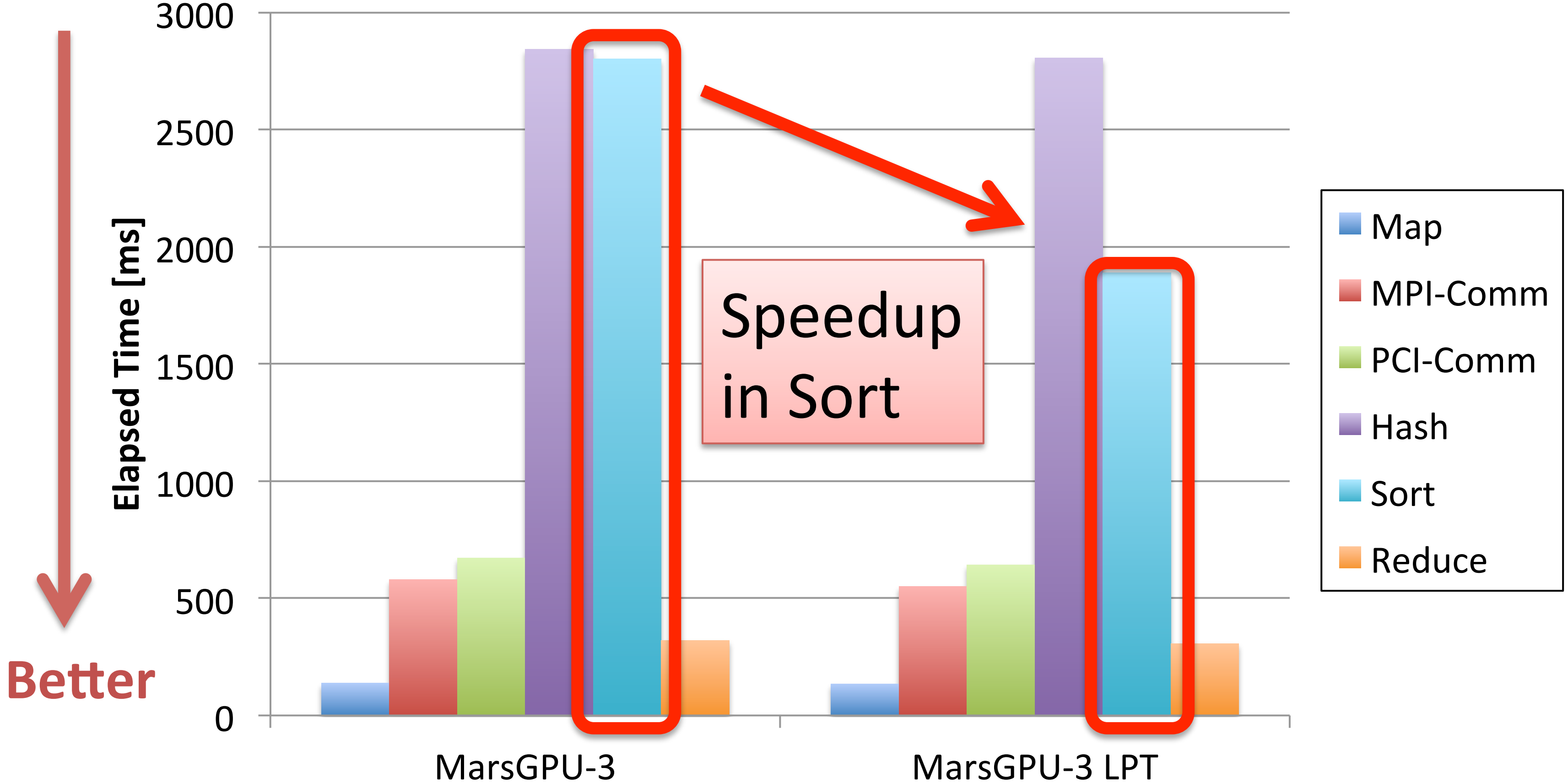
Round Robin vs. LPT Schedule

- Similar except for on 128 nodes
  - Input graphs are relatively well-balanced (Graph500)



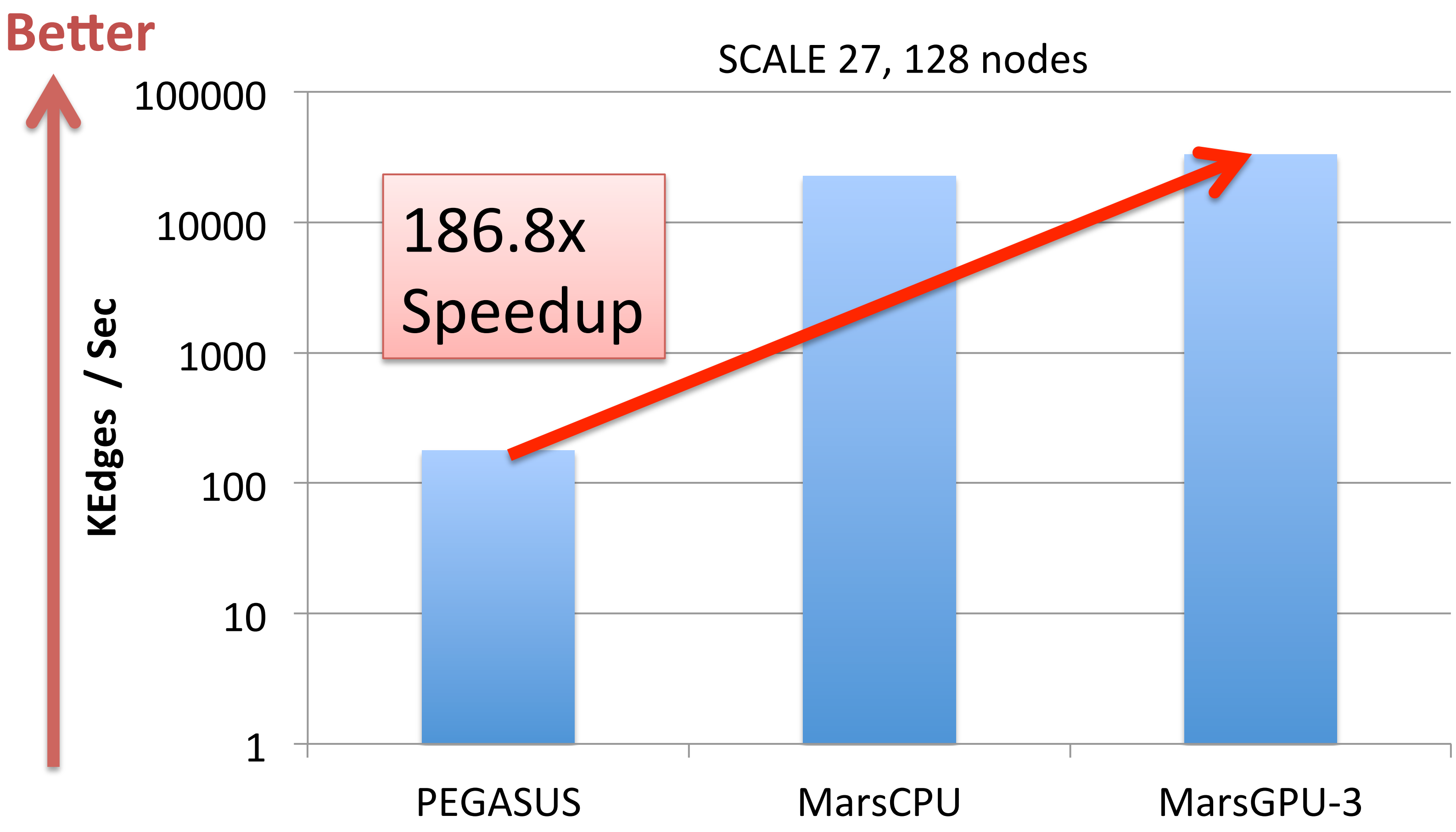
Performance Breakdown
Round robin vs. LPT Schedule

- Bitonic sort calculates power-of-two key-value pairs
  - Load balancing reduced the number of sorting elements



Outperform Hadoop-based Implementation

- PEGASUS: a Hadoop-based GIM-V implementation
  - Hadoop 0.21.0
  - Lustre for underlying Hadoop's file system



Related Work

- Graph processing using GPU
  - Shortest path algorithms for GPU (BFS, SSSP, and APSP)*1

  → Not achieve competitive performance
- MapReduce implementations on GPUs
  - GPMR*2 : MapReduce implementation on multi GPUs

  → Not show scalability for large-scale processing
- Graph processing with load balancing
  - Load balancing while keeping communication low on R-MAT graphs*3

  → We show the task scheduling-based load-balancing

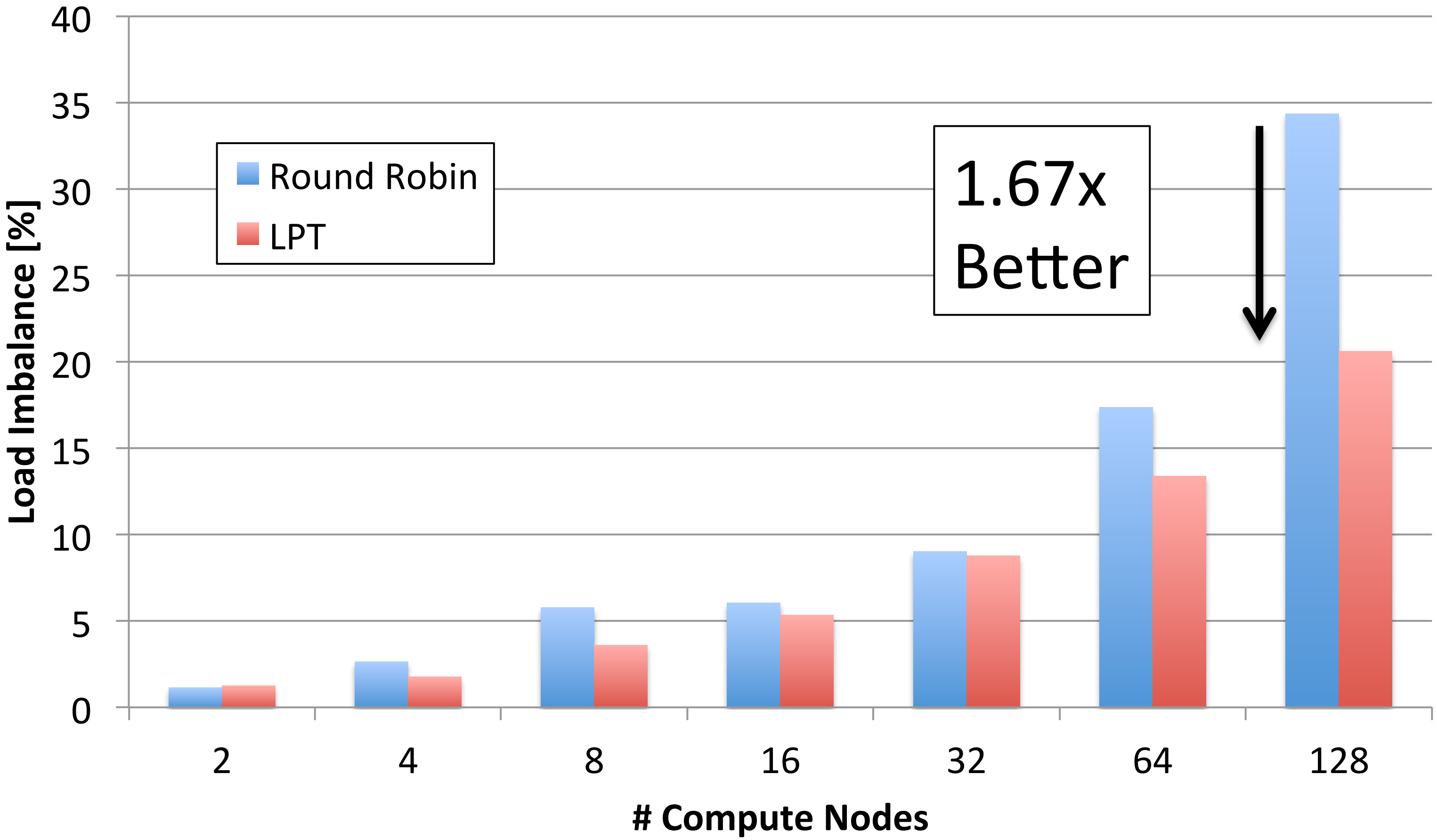
*1 : Harish, P. et al, "Accelerating Large Graph Algorithms on the GPU using CUDA", HiPC 2007.
*2 : Stuart, J.A. et al, "Multi-GPU MapReduce on GPU Clusters", IPDPS 2011.
*3 : J. Chhugani, N. Satish, C. Kim, J. Sewall, and P. Dubey, "Fast and Efficient Graph Traversal Algorithm for CPUs: Maximizing single-node efficiency," in *Parallel Distributed Processing Symposium (IPDPS), 2012*

Conclusions

- **A scalable MapReduce-based GIM-V implementation using multi-GPU**
  - Methodology
    - Extend Mars to support multi-GPU
    - GIM-V using multi-GPU MapReduce
    - Load balance optimization
  - Performance
    - 87.04 ME/s on SCALE 30 (256 nodes, 768 GPUs)
    - 1.52x speedup than the CPU-based implementation
- **Future work**
  - Optimization of our implementation
    - Improve communication, locality
  - Data handling larger than GPU memory capacity
    - Memory hierarchy management (GPU, DRAM, NVM, SSD)

Comparison with Load Balance Algorithm (Simulation, Weak Scaling)

- Compare between naive (Round robin) and load balancing optimization (LPT schedule)
- Similar except for 128 nodes (3.98% on SCALE 25, 64 nodes)
  - Performance improvement: 13.8% (SCALE 26, 128 nodes)



Large-scale Graphs in Real World

- Graphs in real world
  - Health care, SNS, Biology, Electric power grid etc.
  - Millions to trillions of vertices and 100 millions to 100 trillions of edges
  - Similar properties
    - Scale-free (power-low degree distribution)
    - Small diameter
- Kronecker Graph
  - Similar properties as real world graphs
  - Widely used (e.g. the Graph500 benchmark[*1]) since obtained easily by simply applying iterative products on a base matrix

4	3
2	1

→

16	12	12	3
8	4	2	1
8	6	4	3
4	2	2	1

*1 : D. A. Bader et al. The graph500 list. Graph500.org. http://www.graph500.org/